AFRL-RI-RS-TR-2015-210

SECURITY TAGGED ARCHITECTURE CO-DESIGN (STACD)

SEPTEMBER 2015

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2015-210 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
STEVEN T. JOHNS, Chief
Trusted Systems Branch
Computing & Communications Division

/ S /
MARK LINDERMAN
Technical Advisor, Computing &
Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
SEPTEMBER 2015	FINAL TECHNICAL REPORT	JAN 2011 – MAR 2015

4. TITLE AND SUBTITLE

SECURITY TAGGED ARCHITECTURE CO-DESIGN (STACD)

5a. CONTRACT NUMBER
IN-HOUSE

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Jonathan Heiner and Wilmar Sifre

5d. PROJECT NUMBER
T2ST

5e. TASK NUMBER
IN

5f. WORK UNIT NUMBER
HO

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2015-210

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# AABW-2015-4285
Date Cleared: 10 Sep 2015

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Security Tagged Architecture Co-Design (STACD) initiative focuses on eliminating inherent software vulnerabilities by redesigning the underlying hardware and the operating system to enforce software security policies and semantics. The new approach will use a metadata processing unit known as the tagged management unit (TMU) that operates concurrently with the CPU to process the metadata. The introduction of tag capable hardware requires software that uses tagged information. We will develop a tag enabled Operating System (OS) that permits the simplification and reduction in size of the OS for easier verification and validation. The STACD project will co-design a new scalable Security Tagged Multicore Processor (STMP), a Security Tagged Zero-Kernel OS (ST-ZKOS), and a Security Tagged Interconnect (STI) that will maintain metadata through execution without negatively influencing performance by processing the data and its corresponding metadata in parallel. This system will enforce software semantics and security policies, guarantee isolation and separation of information, and provide resistance to malicious attacks. The co-design approach provides a higher assurance of compatibility between the components and a stronger security base.

15. SUBJECT TERMS

Multicore Architecture, Manycore Architecture, Security, Zero-kernel OS, Tagging

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**JONATHAN HEINER**
U	U	U	UU	26	19b. TELEPHONE NUMBER (*Include area code*) **315-330-7750**

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

1.0 SUMMARY

The Security Tagged Architecture Co-Design (STACD) initiative focused on eliminating inherent software vulnerabilities by redesigning the underlying hardware and the operating system to enforce software security policies and semantics.

The new approach uses a metadata processing unit known as the Tagged Management Unit (TMU) that operates concurrently with the Central Processing Unit (CPU) to process the metadata. The introduction of tag capable hardware requires software that uses tagged information. Through contractor collaboration with AFRL/RI personnel, we initiated development of a tag enabled Operating System (OS) that permits the simplification and reduction in size of the OS for easier verification and validation.

The STACD project sought a co-design implementation of a new scalable Security Tagged Multicore Processor (STMP), a Security Tagged Zero-Kernel OS (ST-ZKOS), and a Security Tagged Interconnect (STI) that would maintain metadata through execution without negatively influencing performance by processing the data and its corresponding metadata in parallel. The system sought to enforce software semantics and security policies, guarantee isolation and separation of information, and provide resistance to malicious attacks. The co-design approach provides a higher assurance of compatibility between the components and a stronger security base.

2.0 INTRODUCTION

Today's computer systems are inherently flawed because of outdated hardware/software design choices and practices. Most modern hardware architectures are founded on an architectural design from the 70's. At that time, consumers demanded ever-greater performance and malicious software was practically non-existent. Therefore, OS developers started implementing performance-enhancing techniques that sacrificed many of the security mechanisms implemented by hardware.

Market trends toward enhanced performance lead to an increase in system vulnerabilities, which malicious software can exploit. Malicious software attacks are increasing exponentially and the skill required to implement them is decreasing. The current approach, to patch vulnerabilities when detected has proven to be ineffective. Only by addressing the cause of the vulnerabilities, insecure hardware and software design practices, can we prevent future malicious attacks. This effort sought to co-develop a secure software and hardware platform using secure and proven security principles [1], enforcing software semantics, and following secure design practices.

The major security mechanism implemented in this project is tagging. Tagging is the appending of metadata to instructions or data. The uses of tagging include tracing information flow, providing provenance, or guaranteeing permitted execution. Tagging was one of those security mechanisms developed in the 70's but dropped for enhanced performance due to its expensive memory and execution resource requirement at the time. Since then, lower memory costs, smaller die sizes, enhanced tagging techniques, and a resurgence of security minded engineering encourages the implementation of tagging in modern processors. Current x86 processors implement limited tagging in the form of two-bit privilege levels or single bit information flow tracking at the page level. However, these techniques do not provide enough metadata to enforce software semantics or advanced security policies. Additionally, software does not enforce current

hardware tagging techniques partially due to an emphasis on performance. Taking advantage of new hardware resources, cheaper memory, parallel tag execution, and a new OS design, this work incorporates tagging into a system's architecture with a goal of incurring less than a 2-3x performance decrease.

The following section describes previous research related to this effort, followed by concurrent research efforts that utilize security tagging.

2.1 Security Tagging Background

Security tagging is not new; several processors implement tagging. The Burroughs-Unisys MCP/AS system [2, 3] enabled security-tagged data in their processor. They required that all code for the system be in a high-level language and compiled on their compiler. However, due to performance concerns, this requirement was not enforced and allowed the use of legacy code that violated their security guarantees.

IBM's System 38 [4] allows users to use legacy code and increase information assurance by implementing a single bit of tagging to distinguish data and instructions from capabilities. A capability is a data structure that contains the access rights of an object, and a pointer to that object. An object is a physical entity of the system, such as a memory segment, a register, or a peripheral. The access rights describe how the object can be used. IBM's System 38 provides users with the option to extend the security of their system by using a single bit for tagging. Using a single bit is insufficient to allow for multiple levels of security (MLS). A single bit only represents two levels of separation. Conversely, Shioya et al, [5], describe a variable length tag technique which they employ to reduce the hardware overhead of a fixed length tagging scheme. However, they assume that the system will tag only a subset of the information. This approach limits the uses of tagging to tracking information flow only and not for security. It is important to note that information flow tracking can be used to enhance the security of the system; however, depending on how the tags propagate through the execution flow and how the system reacts to handling of the tags determines on whether it benefits the security of the system or not.

Current processors, such as Intel's x86 architecture, extend tagging from a single bit to two bits. These two bits provide what is known as the ring architecture [6], as shown in Figure 1. The ring architecture separates information into three domains; 0 – The Kernel Domain, 1 & 2 – Middle Domains (largely ignored), 3 – User Domain. All kernel code and data must operate in the Kernel Domain (ring 0) while user code and data must remain in User Domain (ring 3). This technique increases the security of the system by providing isolation and separation of information, adhering to the security policy.

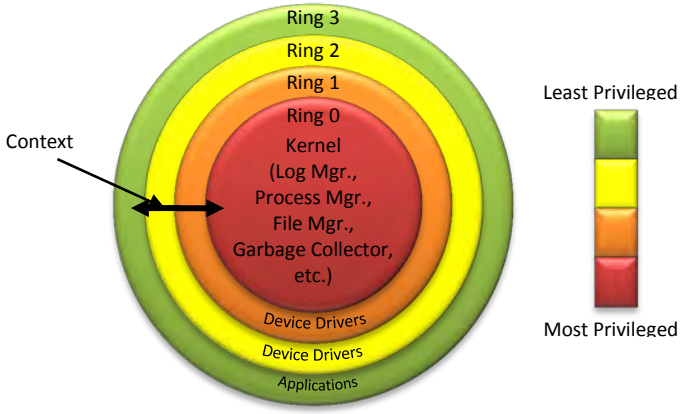

Figure 1: Ring Architecture. Lower-level rings have more privileges than higher-level rings. A lower level ring component can access any higher-level ring space. Any component in a given ring level has access to all components in the same ring level whether needed or not. To move from one ring to another a context switch is required.

However, they did not take into consideration the systems software. In order to use certain system functions, the user must perform a costly context switch into the Kernel Domain. OS developers realized that application developers and users would not accept this costly performance hit so they allowed software to violate the security policy setup by the hardware. Users could then inject a portion of their code into the Kernel Domain, allowing unlimited access to the rest of the system and anyone else's code, see Figure 2. Consequently, the kernel grew to a size where verification and validation is impractical [7].

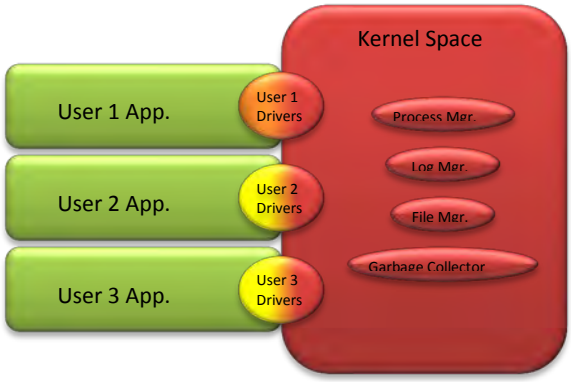

Figure 2: Modern monolithic kernel. Users are allowed to place "trusted" or "essential" components of their software in the kernel space to alleviate the high costs of context switches. However, once in kernel space, those components have access to all other system components whether they need it or not.

Microkernels [8, 9, 10] seek to reduce the kernel size to improve performance. Other software approaches include virtualization and hypervisors. Virtualization and hypervisors [11, 12, 13] inject an additional layer of abstraction between the processor and the systems software. These techniques allow the system to separate applications into specific domains to prevent a malicious application from interfering with other system operations. However, this approach causes significant performance overhead, and does not provide the fine-grained control to allow for formal verification.

Microkernels reduce the size of the kernel enough to allow for a formal verification of the kernel. Tanenbaum [14] documents many of the security virtues of microkernels and argues that the

advantages of such a system may be worth the performance impact. However, these systems still rely on a centralized privileged unit. With such a unit available, programmers will still violate software semantics and design policies by integrating their code into the microkernel to improve their applications performance, thus voiding any security gains achieved.

In order to address the limitations presented by current processors, virtualization, and microkernels, the Intelligence Advance Research Projects Agency (IARPA) funded a research project entitled, "Trust-management, Intrusion-tolerance, Accountability, and Reconstitution Architecture. [15]" This project started the initial concept of a hardware/software co-design for tagged architectures. The research focused on emulating a single core tagged processor and a Zero-Kernel OS (ZKOS). The single core tag-enabled processor introduced a tag management unit that is capable of maintaining metadata through execution. Their ZKOS continued the trend of reducing the size of the kernel for security purposes by dividing the independent functions of the kernel into separate components. These components reside in their own isolated memory space and adhere to individually tailored security policies while eliminating the need for user and kernel separation, see Figure 3. By eliminating user and kernel separation, they replaced the expensive context switches with less expensive component procedural calls. This research has shown that tag-enabled processor architectures with a zero-kernel OS provides significant security benefits while theoretically providing a limited impact on performance. The amount of work provided under the IARPA contract showed the proof of concept but did not extend the work to multicore architectures or Multiple Independent Levels of Security (MILS) applications, nor provides a complete ZKOS.

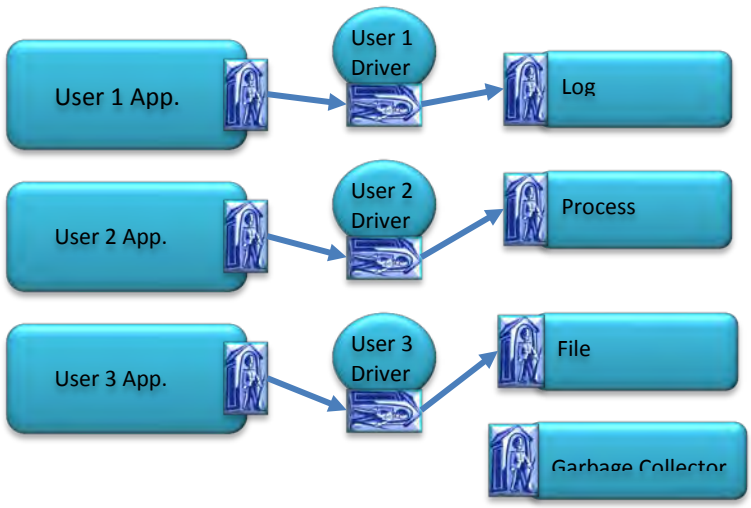

Figure 3: ZKOS architecture. Each components of the software resides in its own isolated memory space. Guards guarantee that a user's component will only access other system components that it needs to access, based on tag information, and will prevent access to all other components, per the security policy.

In this project, we co-developed a secure ZKOS that is capable of using MILS applications, a tagged secure multicore processor that is capable of processing the many-bit metadata provided by the OS, and a tagged interconnect system that is capable of transmitting and temporarily storing the metadata before and after execution. Research towards the secure ZKOS and the multicore processor were performed under contracts while in-house research provided the research towards the interconnect system. We developed a framework for co-developing the hardware and software necessary for secure systems. We identified platforms for future secure

OS, core, and interconnect development. The research resulted in many novel concepts in OS design, processor architectures, cache hierarchies, and interconnect infrastructures.

2.2 Concurrent Security Tagged Research

Concurrently in 2011 the Defense Advanced Research Projects Agency (DARPA) kicked-off their Clean-Slate Design of Resilient, Adaptive, Secure Hosts (CRASH) program. DARPA CRASH sought to pursue innovative research into the design of new computer systems that are highly resistant to cyber-attack, can adapt after a successful attack to continue rendering useful services, learn from previous attacks on how to guard against and cope with future attacks, and can repair themselves after attacks have succeeded. [16]

Performers under the DARPA CRASH program were to develop a novel architecture, based on security. This clean-slate approach involved the co-design of hardware, system software, programming languages, design environments, and formal methods. Some of the performers approached the CRASH program similar to the STACD project; that is, to investigate a co-design of the hardware and software with large scale tagging (greater than 8-bit tag per 32-bit word) and ZKOS-like OSs. [17, 18]

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

Three research organizations collaborated during the STACD project; AFRL/RI, Cornell University, and the University of Idaho. For completeness and reference, we will discuss the research performed by the University teams, along with the in-house research project. The University of Idaho, who worked towards the Security Tagged Zero-Kernel OS research, was under contract FA8750-11-2-0047 and Cornell University, who worked towards the Security Tagged Multicore Processor, was under contract FA8750-11-2-0025. AFRL/RI in-house researchers performed the Security Tagged Interconnect research.

3.1 Security Tagged ZKOS (Idaho)

Under this task, the operating system team sought to develop a new operating system architecture in the presence of a new security tagged multicore processor architecture. The new OS architecture sought to enhance the features of the STMP to enforce the data isolation and information flow properties of MILS, extend the capabilities found in current STMP research by extending the focus away from detecting and preventing malicious code and towards providing a stable platform to assist software developers in creating more secure code, and provide software developers with a set of tools and features to enable them to specify and enforce application-level security policies.

The operating system team collaborated with the hardware team in analyzing the security policies and support mechanisms required for a STA. Based upon their analysis, the team developed an OS architecture that fully utilizes the MILS principles of separation and controlled information flow, providing software developers with OS support for the development and enforcement of application-level security policies.

The software team sought to develop the OS to be compatible with legacy applications as well as legacy processors. The OS would support the ability to develop wrappers, virtual machines and middleware that can utilize the new security features and architecture to enhance the security of

the legacy applications. The OS sought to extend the basic MILS model to support the deployment of applications across multiple cores within a single processor package. However, as will be discussed in the Results and Discussion section, the development of the ST-ZKOS is incomplete.

3.2 Security Tagged Multicore Processor (Cornell)

The goal of this task was to realize the full potential of fine-grained tagging by developing a flexible tagged architecture for multicore platforms that can efficiently support many tagging techniques including zero-kernel operating systems. The processor team realized a broad range of tagged architecture designs by tightly coupling an on-chip reconfigurable fabric, such as a Field Programmable Gate Array (FPGA), with a static processing core, see Figure 4. Coupling the reconfigurable fabric with the static processing core allows the evolution of the tagged architectures even after fabrication. This architecture enables the tagged architecture design to be co-developed with software components, such as zero-kernel operating systems, and to easily evolve as the software needs change. In addition, the flexible architecture allows a system to be upgraded with a new tagging technique or patched to fix hardware vulnerabilities in the field. The reconfigurability also provides a promising way to add tagged architecture functions to Commercial Off the Shelf (COTS) processors.

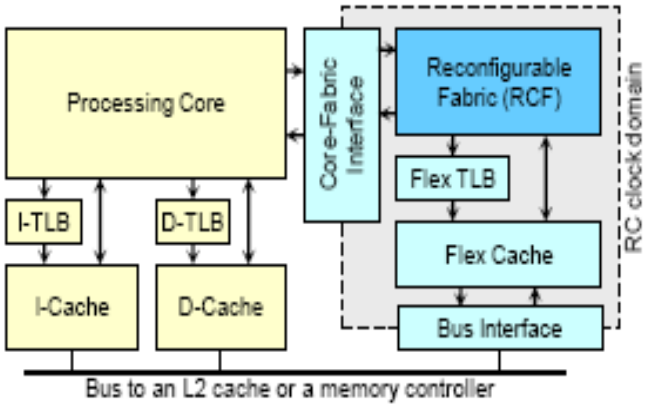

Figure 4: STMP Architecture. Each core of the multicore system is directly connected to an associated Flex-core RCF logic that performs tag checking techniques in parallel to core execution.

The processor team completed one of three phases in the development of the tagged multicore processor. The first phase involved close interactions between the processor team and the operating system development team to define the security tag schemes that is used in the OS and how the architecture will support them. Upon completion of the architecture design, the processor team developed a tagged single core prototype on an FPGA. The processor team provided the FPGA implementation to the OS team for software development.

After the development of a tagged single core architecture, the second and third phases consist of two major technical components that are largely independent from each other. First, the processor team sought to develop a programming infrastructure to enable easy and fast customization of the tagging hardware. The goal of this infrastructure is to make generating hardware for security tags as easy as writing a C program. Second, they sought to extend the

single core architecture to multicore platforms with support for multi-threaded applications. This extension will include extending the FPGA prototype system with multiple cores. The STMP team will provide both the programming infrastructure and the FPGA system to the ST-ZKOS team for the investigation and implementation of new tagging techniques and multicore OS extensions. However, as will be discussed in the Results and Discussion section these last two phases were not completed.

3.3 Security Tagged Interconnect System

The goal of this task is to extend the capabilities of the STMP and ST-ZKOS by providing a security tagged interconnect system covering the hardware components necessary for OS and processor communications. Examples of these components are: communications bus, cache hierarchy, and memory controller.

Previous research in tagging techniques do not address the interconnect system since tagging is only implemented in the processor core or embedded in the cache line. Therefore, this area of research is largely overlooked and incomplete. However, if metadata is present in the OS and needs to be presented to the processor, the interconnect system has to be capable of transmitting and temporarily storing the metadata with its associated data.

To develop the interconnect system, the interconnect team sought to develop a Hardware Description Language (HDL) model of a current interconnect system. The OS team and processor team provided the interconnect team with a description of their chosen preliminary tagging techniques. The tagging techniques indicate the number of bits required for the metadata, and the temporary storage algorithm needed. The interconnect team used this information to expand the interconnect system model to incorporate the tagging requirements. The interconnect team sought incorporate additional tag-based security policies to enforce strict isolation on communication mediums and in temporary storage locations.

3.4 Tasks

3.4.1 Security Tagged ZKOS (Idaho)

The ST-ZKOS team sought to develop a novel OS for multicore architecture with tagging capabilities. The design of this OS replaces costly context switches with more efficient procedural calls. Additionally, when combined with adequately tag-enabled hardware, each component of the OS can meet MILS and security policy requirements by providing complete isolation of information. This type of architecture not only addresses today's current threats, but additionally eliminates future threats by addressing the root causes of security violations. The following tasks were the original task set for the operating system development team:

- Security Policies: Investigate the set of security policies that are enforceable by a STA. The team will investigate if a STA can support the set of all run-time enforceable security policies. We contend that all executable hardware of the system (DMA controllers, co-processors, network cards, etc.) will have to conform to STA principles, or will have to be isolated by STA hardware in order to provide strong assurance of run-time enforceable security policies.

- Architectural Design: Develop the architecture of a new class of operating system that utilizes security enforcement mechanisms of tagged-architectures to implement a MILS compliant system.
- OS Prototype: Develop a prototype operating system that enforces the security policies identified in the first task through compartmentalization of the OS and information tagging. The OS's target platform is the STA hardware architecture co-developed by Cornell University and AFRL/RI.

3.4.2 Security Tagged Multicore Processor (Cornell)

The STMP sought develop a novel multicore processor architecture that is capable of providing the hardware support for a security tagged OS. This type of architecture can be used to augment modern COTS systems or provide the framework for future AF secure processors. By designing the processor in conjunction with the OS we provide the greatest possible level of security by eliminating vulnerabilities that can be introduced by third-party middleware developers. The original tasks for this area of research are as follows:

- Single core Architecture: Study various security tagging mechanisms and develop a generic model supported by the reconfigurable tagged architecture. Design the reconfigurable tagged architecture for a single core embedded processor, a single-issue in-order processor with a moderate clock frequency. Extend the tagged architecture for processors with high clock frequencies and/or superscalar mechanisms. Develop mechanisms to ensure the security of the reconfigurable tagging mechanism: trusted programming and isolation of reconfigurable hardware. Implement the tagged architecture and selected extensions in Register Transfer Level (RTL). Evaluate the performance, area, and power consumption.
- Multicore Architecture: Extend the tagged architecture to handle multi-threaded applications on multicore systems. Investigate the sharing of tagging hardware among multiple processing cores. Implement the multicore extensions in RTL. Evaluate the performance, area, and power consumption.
- Programming Model: Develop a programming model and an infrastructure to program the reconfigurable tagging hardware in a high-level language (C/C++). Implement a set of tagging mechanisms in a high-level language (C/C++) and evaluate them compared to RTL implementations. Optimize the reconfigurable fabric architecture and the high-level programming infrastructure to closely match the efficiency of low-level RTL implementations.
- FPGA Prototype: Implement a single core tagged architecture and selected tagging mechanisms with the Leon3 (or similar) FPGA processors. Implement a multicore tagged architecture with two or more cores on the FPGA system.
- CAD Infrastructure: Setup a standard-cell Application-Specific Integrated Circuit (ASIC) flow for the IBM 65nm process. Develop a Computer Aided Design (CAD) flow for the reconfigurable fabric. Develop the area, speed, power models for the fabric. Setup a high-level synthesis tool (C/C++ to HDL).

3.4.3 Security Tagged Interconnect

The STI team sought to develop a novel interconnect subsystem to a STMP architecture. This interconnect is essential in providing the tagging support between software and hardware. This research also identifies a means of providing enhanced interconnect security to systems that do not employ tagging mechanisms between the hardware and software. The original tasks for this area of research are as follows:

- Research Architectures: Identify interconnect architectures and desired capabilities. Identify security policies for interaction between CPU and OS.
- Design Interconnect: Develop architecture for interconnect.
- Interconnect Prototype: Develop or modify VHSIC Hardware Description Language (VHDL) model for interconnect systems.
- Interconnect Security: Integrate security tagging techniques into interconnect design for both single core and multicore architectures.
- Project Cohesion: Prototype entire system by integrating all components developed during lifetime of the project (STMP, STI, and ST-ZKOS).

3.5 Metrics

3.5.1 Security Tagged Zero-Kernel OS

- Performance Metric: 25% fewer context switches. No comparable data from previous techniques due to novel approach.
- Size Metric: 10% fewer Lines of Code (LoC) compared to similar OS, 10% fewer trusted Application Programming Interface (API) instructions than similar OS.
- Security Metric: 100% detection/prevention of half of 2011 CWE/SANS Top 25 Most Dangerous Software Errors [16], see APPENDIX A for list of targeted errors.
- Completeness Criteria: Functioning OS, i.e. correct execution and separation of applications, capable of interfacing with tagged hardware.

3.5.2 Security Tagged Multicore Processor

- Performance Metric: A 2X to 3X performance degradation using tagging to implement Dynamic Instruction Flow Tracking (DIFT) compared to normal operation. Previous DIFT techniques incurred a performance degradation of 3.6X to 37X.
- < 10% performance degradation compared to software running on CPU alone.
- Area Metric: < 20% increase in area compared to core. No comparable data from previous techniques due to novel approach.
- Security Metric: 100% detection/prevention of half of 2011 CWE/SANS Top 25 Most Dangerous Software Errors [16], see APPENDIX A for list of targeted errors.
- Completeness Criteria: Maintains correctness of tags through execution. Determined through analysis of input and output compared to a golden copy.

3.5.3 Security Tagged Interconnect

- Performance Metric: < 10% performance degradation through interconnect. No comparable data from previous techniques due to novel approach.
- Area Metric: < 100% increase in area compared to original interconnect system.
- Completeness Criteria: Correct transfer of data and associated tag information. Verified through simulation and testing.

4.0 RESULTS AND DISCUSSION

This project is being closed while the research is incomplete. The in-house research team's responsibilities and research objectives were redirected during the execution of the research. The following paragraphs are descriptions of the state of the program at the time of closure. Afterwards we will describe future work that needs to be addressed for successful completion of the project.

4.1 Security Tagged ZKOS (University of Idaho)

4.1.1 Security Policies (Task 1):

The University of Idaho worked closely in collaboration with AFRL and Cornell University in the development of the security policy. The generated security policy enforced proper memory access, control flow operation and machine code operations. Greater detail of security policy can be found in [20]. Because of the level of control required by the software component that assigns the tag values the additional complexity was introduced into the OS design which will require increased verification and validation.

The ST-ZKOS implements a 32-bit tag that is paired with each 32-bit word in memory. This effectively cuts the amount of memory. Figure 5 depicts the chosen format for the 32-bit tag. There are three primary fields [20]:

1. Owner Field – this field indicates the entity that owns the resource managed by the code module. All code and data on the system have been separated into code modules that perform specific functions based on the concept of least privilege. An example of a code module would be the garbage collector, or a device driver.
2. Code-space Field – this field indicates the code modules that are currently executing and/or the code modules that are authorized to access specific operating system resources.
3. Control-bits Field – this field is used to even further support least privilege by providing some typing and access control information to system resources.

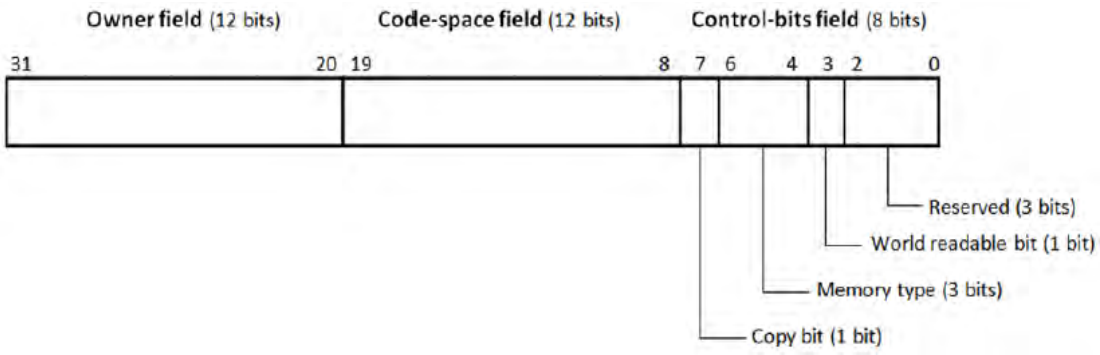

Figure 5: ST-ZKOS Tag Format

4.1.2 Architectural Design (Task 2):

The original concept for the ST-ZKOS was to develop the OS from scratch. However, due to other research efforts at AFRL it was determined to instead modify the Real-Time Executive for Multiprocessor Systems (RTEMS). Starting with RTEMS allowed the ST-ZKOS team to focus on the system architecture and the security aspects of the OS core components. Additionally, because RTEMS has no concept of user, and each component of RTEMS is already isolated, this fit well within the design concept for a ZKOS.

Through collaboration with Cornell University and AFRL, the ST-ZKOS team was able to identify the architectural changes to RTEMS that were necessary in order to implement the security policy chosen in Task 1. Further detail on these changes can be found in [20].

4.1.3 OS Prototype (Task 3):

This task was to implement the architectural changes required to modify RTEMS into a ST-ZKOS. The first spiral integrated the security tags throughout the architecture of RTEMS and to place the specific hooks and traps for hardware collaboration. Through this spiral the ST-ZKOS and STMP teams swapped models and simulations of their designs to ensure compatibility.

The second spiral was to enhance the ST-ZKOS to include multiple user and multiprocessor support. However, this portion of the research is incomplete.

4.2 Security Tagged Multicore Processor (Cornell University)

4.2.1 Single-core Architecture

The STMP is based on Cornell's FlexCore architecture [20], a modified Leon3 OpenSparc open source processor with a coupled on-die FPGA fabric that provides the reconfigurable tag management capability. The STMP was modified to implement a security tagging scheme co-developed with the University of Idaho and AFRL. Additional modifications were made to the processor to include concessions for real-time execution, such as tag cache compression, additional tagging schemes, and high-performance tag management. Further detail on the STMP design can be found in [22]. However, precise exception handling was not implemented.

4.2.2 Multi-core Architecture

Upon completion of the single-core processor version of the STMP, Cornell University was to further modify the STMP design to include support for multiple cores. Initial progress was made towards this end with the development of a dual-core configuration of the unmodified Leon3 processor. However, this research is incomplete.

4.2.3 Programming Model

Many hardware-based security features are often unused due to the complexities for software developers in using them. This task sought to provide a high-level synthesis tool to allow a software developer to implement custom tagging schemes in the hardware. Initial investigation in this area looked towards using C-to-Silicon, a Cadence high-level synthesis tool, to implement tagging schemes written in SystemC [23]. However, this research was unable to be pursued further.

4.2.4 FPGA Prototype

A single core version of the STMP without precise exceptions has been completed and provided to all teams. A demonstration of the hardware architecture, not with the ST-ZKOS was completed to demonstrate the capability of using the hardware-based tags for security. However, thorough testing and integration with the ST-ZKOS or STI are incomplete. Further detail of the FPGA prototype and initial testing can be found in [22].

4.2.5 CAD Infrastructure

This research is incomplete.

4.3 Security Tagged Interconnect (In-House)

4.3.1 Research Architectures

The STI team reviewed open literature on security-based tagging, the LEON3 OpenSparc processor, and the ARM AMBA bus.

4.3.2 Design Interconnect

Figure 6 shows the high level block diagram of the Leon3 architecture [24]. The Leon3 OpenSparc processor utilizes ARM's AMBA Bus for communication to all on-chip peripherals. System components connect to the AMBA bus as either Master or Slave. A component may be labeled as both master and slave, but require separate ports for each designation. The AHB controller uses a bus arbiter, multiplexer, and decoder to control the flow of information on the bus. As each component is added to the bus, they exchange plug & play configuration information with the AHB controller that identifies the provenance of the component, and any addressing requirements. Figure 7 shows the layout of the configuration record.

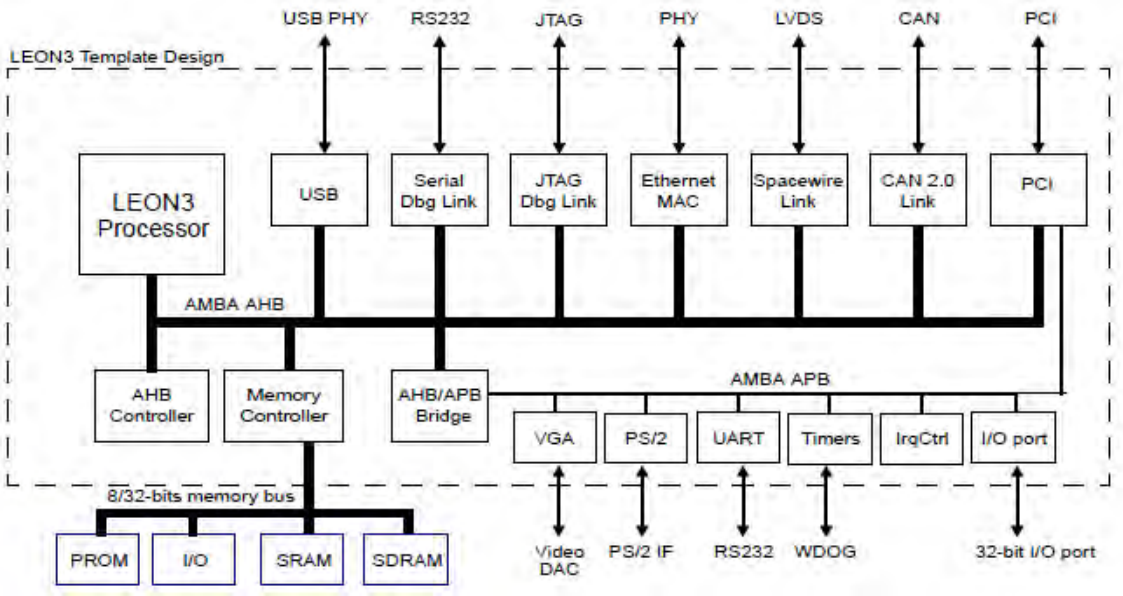

Figure 6: Example Leon3 system architecture

The STI team sought to modify AMBA Bus architecture such that each component on the bus would be associated with a single owner at any given time. Any master component owned by one entity would not be able to read/write from/to any slave component owned by another entity. Additionally, as we cannot guarantee the provenance of all components and the intent of their designers, we sought to prevent any component from accessing the bus, except for access requests, until given permission by the controller. The current architecture publishes the data to all components and indicates which component should read the data. Permitting all components open access to read the bus allows for a potentially malicious peripheral from retrieving and potentially transmitting the information. To accomplish these two goals we included the tagging scheme in the AMBA architecture. The final modification includes providing a means for the tag associated with code/data to be moved about the system with the data.

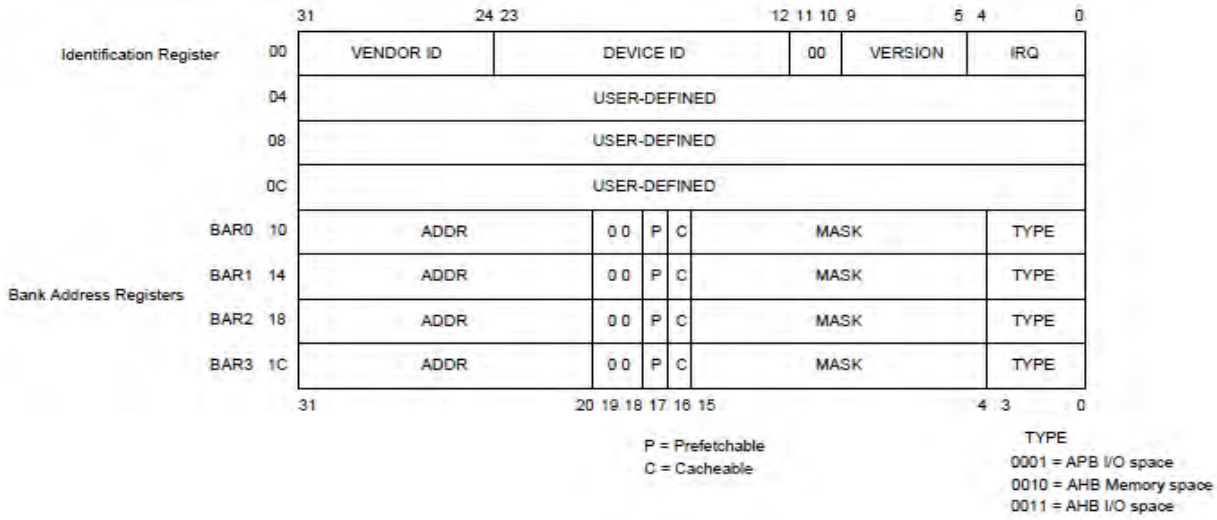

Figure 7: AHB plug & play configuration layout

There are two options for isolating each component on the AMBA bus: 1. Create a wrapper for each component that guarantees that each component can only view the data when given permission. 2. Modify the bus architecture and arbiter such that each component has a dedicated line from the arbiter and that only that line is updated as needed.

As it turns out the AMBA architecture is not completely a bus. Outputs from the AHB controller to the components are bus wires that connect to all components. However, inputs to the AHB controller from the components are record arrays. Each record in the array corresponds to a specific component. Therefore, we chose to implement the second option as it would require less modification. The modifications required extending the outputs from the AHB controller to the components to be record arrays, where each element in the array is dedicated for each component, and an additional mux at the end of the AHB controller to indicate which line(s) to put the data on. Of course this will increase the area size, how much has yet to be determined.

As code and data move around the system it is important to maintain their respective tag value. Maintaining the tag with its associated code and data can be dealt with in two ways: 1. Extend the path the data/code takes to include the tag, i.e., a separate AHB AMBA bus that moves the tag when code/data is moved. 2. Stagger code/data and tag reads/writes. The first option increases the overall space of the processor as now we need two ports to each component, one from each bus, and a completely new bus. However, this approach is the simplest. The second option maintains the size of the processor but includes possible performance degradations since it requires two reads/writes per operation. For example if the Leon3 core seeks to write data to memory, first the tag of the data needs to be written to the memory location and then the data can be written. For reads the tag must be read first and then the data. However, the AMBA bus would identify the two operations separately and may not allow the data to follow directly after the tag if another master with higher priority is granted access. Then, the receiving component must initiate a split operation. This requires wrappers on each component. Due to the complexity of this solution we opted to pursue the first option, that of doubling the data path to compensate for the tags.

In order to associate each component on the bus with a specific owner, the components needed a way to identify who their owner is. The owner field of the tagging scheme developed for the ST-ZKOS and STMP allows each component to identify an owner. The other fields of the tag are used by the tag management unit to indicate what rules the data/code must follow in the processor and are not relevant for the interconnect. However, to not eliminate potentially useful information we maintained the whole tag format, shown in Figure 5, for each component.

Software needs a means to set the tag value for each component, thus identifying the owner. To accomplish this, it was identified that the AHB plug & play information for each component is stored in a record array in the arbiter of the AHB controller. The arbiter needs to be modified such that this array is now memory mapped so that software can address it to assign tags for each component. When a master component, after having been granted sole access to the bus, writes data to a specific address, the arbiter will interpret the address to identify which slave component should receive the data, and will also compare the tag of the master with the tag of the slave from the memory mapped array to determine if they are owned by the same entity. If they are not, then the arbiter reports an error and cancels the transaction.

The only caveat to the previous approach is with memory components. Most memory components are shared among various owners. Software needs to ensure that one owner does not

attempt to overwrite the memory locations of another owner. The arbiter will not perform tag checks on writes to memory, such as Direct Memory Access (DMA) writes. For DMA writes, the arbiter will assign the master's tag to all data from the master on the tag bus to memory. This approach does not sacrifice security as the new data is tagged appropriately according to the owner of the master. Therefore, it is important that software assign the tag appropriately. The arbiter will perform tag checks on reads from memory when the requesting master is not a processor. If the requesting master is not a processor then the tag of the data is compared to the tag of the requesting master. If the tags do not match then the arbiters initiates an error response and terminates the transaction.

4.3.3 Interconnect Prototype

Upon delivery of the single-core version of the STMP, the STI team sought to modify the AMBA architecture according to the design developed in the previous task. However, the contracts supporting the STMP team were extended, requiring less time dedicated towards the development of the STMP during the second year of research. This in turn required the STI team to take on additional research activities and responsibilities. Shortly after the STMP team finished the development of the single core version of the STMP, both the STMP and ST-ZKOS projects were closed. Due to the lack of further support from the STMP and ST-ZKOS team, the STI team ceased research towards the prototype, taking up other research activities and responsibilities.

4.3.4 Interconnect Security

See comments from the Interconnect Prototype task.

4.3.5 Project Cohesion

The STI team facilitated a close collaboration between the University of Idaho and Cornell University, allowing for an easier integration of components. Currently, the STMP prototyped by Cornell University implements the security policies defined by the University of Idaho's ST-ZKOS. Cornell University has shared the prototype code with AFRL/RITA and the University of Idaho. However, no further research has been performed.

5.0 CONCLUSION

The goal of the project was to develop a Security Tagged System based on a Security Tagged Zero Kernel Operating System, a Security Tagged Multicore Processor, and a Security Tagged Interconnect. Three teams collaborated in the development of novel tagging schemes that guarantees isolation and enforces a security policy of least privileges. The STMP team demonstrated the tagging scheme on an FPGA emulation using a Leon3 soft-core open source processor. The ST-ZKOS team modified RTEMS to implement the tagging scheme. The STI team developed the techniques necessary to modify the AMBA AHB bus for tag propagation, tag management, and component isolation. Unfortunately, due to shifting responsibilities the research is incomplete and not fully implemented or tested.

However, this research has generated interest in tag-based security within other research programs at AFRL. The Robust and Secure Processor Project will seek to implement a less robust form of tagging in their next generation processor design with the goal of increasing the difficulty for remote malicious attackers from gaining a foothold on the system through code injection, one of the benefits of a tagged architecture.

6.0 RECOMMENDATIONS

Industry has seen that tag-based security features provide significant benefits with limited overhead. Examples are Intel's hardware extensions, ARM's TrustZone, and DARPA's CRASH Program. It is recommended that this research continue.

REFERENCES

[1] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems.," *Proceedings of IEEE,* vol. 63(9), pp. 1278-1308, 1975.

[2] Intel Corporation, Introduction to the iAPX 432 Architecture Manual, Santa Clara, CA, 1981.

[3] E. I. Organick, Computer System Organization: The B5700/B6700 Series, Academic Press, 1973.

[4] H. M. Levy, Capability-based Computer Systems, Bedford, MA: Digital Press., 1984.

[5] R. Shioya, D. Kim, K. Horio, M. Goshima and S. Sakai, "Low-Overhead Architecture for Security Tag," *Pacific Rim International Symposium on Dependable Computing, IEEE,* vol. 0, no. 0, pp. 135-142, November 2009.

[6] Intel Corporation, Intel 64 and IA-32 Architectures Software Developer's Manual, Santa Clara, 2010.

[7] G. Heiser, "Secure Embedded Systems need Microkernels," *USENIX ;login,* vol. 30, no. 6, pp. 9-13, December 2005.

[8] D. Golub, R. Dean, A. Forin and R. Rashid, "Unix as an application program.," in *Proceedings of the Summer 1990 USENIX Conference*, 1990.

[9] J. N. Herder, "Towards a true microkernel operating system," 2005.

[10] M. D. Schroeder, D. D. Clark, J. H. Saltzer and D. Wells, "Final Report of the Multics Kernel Design Project," Massachusetts Institute of Technology, 1978.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the Art of virtualization," in *Proc. 19th ACM SOSP*, 2003.

[12] R. J. Creasy, "The origin of the VM/370 Time-sharing System," *IBM Journal of research and Development,* vol. 25, no. 5, pp. 483-490, September 1981.

[13] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," *IEEE Computer,* vol. 38, pp. 39-47, 2005.

[14] A. S. Tanenbaum, J. N. Herder and H. Bos, "Can we make operating systems reliable and secure?," *Computer,* vol. 39, no. 5, pp. 44-51, May 2006.

[15] H. Shrobe, A. DeHon and T. Knight, "Trust-management, Intrusion-tolerance, Accountability, and Reconstitution Architecture," AFRL/RITA, 2009.

[16] DARPA, "Clean-Slate Design of Resilient, Adaptive, Secure Hosts (CRASH)," DARPA, 2010. [Online]. Available: http://www.darpa.mil/Our_Work/I2O/Programs/Clean-slate_design_of_Resilient_Adaptive_Secure_Hosts_(CRASH).aspx. [Accessed 2010].

[17] S. Chiricescu, A. DeHon, D. Demange, S. Iyer, A. Kliger, G. Morrisett, B. C. Pierce, H. Reubenstein, J. M. Smith, G. T. Sullivan, A. Thomas, J. Tov, C. M. White and D.

Wittenberg, "SAFE: A Clean-Slate Architecture for Secure Systems," in *IEEE International Conference on Technologies for Homeland Security (HST)*, Waltham, MA, November 2013.

[18] R. N. M. Watson, J. Woodruff, P. G. Neumann, S. W. Moore, J. Anderson, D. Chisnall, N. Dave, B. Davis, K. Gudka, B. Laurie, S. J. Murdoch, R. Norton, M. Roe, S. Son and M. Vadera, "CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization," in *36th IEEE Symposium on Security and Privacy*, San Jose, CA, 2015.

[19] B. Martin, M. Brown, A. Paller, D. Kirby and S. Christey, "2011 CWE/SANS Top 25 Most Dangerous Software Errors," MITRE Corporation, 2011.

[20] J. Alves-Foss, J. Song, S. Steiner and S. Zakeri, "A New Operating System for Security Tagged Architecture Hardware in Support of Multiple Independent Levels of Security (MILS) Compliant Systems," DTIC, 2014.

[21] D. Y. Deng, D. Lo, G. Malysa, S. Schneider and G. E. Suh, "Flexible and Efficient Instruction-Grained Run-Time Monitoring Using On-Chip Reconfigurable Fabric," in *Proceedings of the 43rd International Symposium on Microarchitecture*, 2010.

[22] G. E. Suh, "Flexible Tagged Architecture for Trustworthy Multi-Core Platforms," DTIC, 2015.

[23] M. Ismail and G. E. Suh, "Fast Development of Hardware-Based Run-Time Monitors Through Architecture Framework and High-Level Synthesis," in *Proceedings of the 30th International Conference on Computer Design (ICCD)*, October 2012.

[24] Gaisler Research, "GRLIB IP Library User's Manual Version 1.1.0 B4100," Gisler Research, 2010.

[25] A. Azevedo de Amorim, N. Collins, A. DeHon, D. Demange, C. Hritcu, D. Pichardie, B. C. Pierce, R. Pollack and A. Tolmach, "A Verified Information-Flow Architecture," in *41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, San Diego, 2014.

APPENDIX A

Rank	ID	Name	Target for Coverage
[1]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	Y
[2]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	Y
[3]	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	Y
[4]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	Y
[5]	CWE-306	Missing Authentication for Critical Function	Y
[6]	CWE-862	Missing Authorization	Y
[7]	CWE-798	Use of Hard-coded Credentials	N
[8]	CWE-311	Missing Encryption of Sensitive Data	N
[9]	CWE-434	Unrestricted Upload of File with Dangerous Type	Y
[10]	CWE-807	Reliance on Untrusted Inputs in a Security Decision	Unsure
[11]	CWE-250	Execution with Unnecessary Privileges	Y
[12]	CWE-352	Cross-Site Request Forgery (CSRF)	N
[13]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	Y
[14]	CWE-494	Download of Code Without Integrity Check	N
[15]	CWE-863	Incorrect Authorization	Y
[16]	CWE-829	Inclusion of Functionality from Untrusted Control Sphere	Y
[17]	CWE-732	Incorrect Permission Assignment for Critical Resource	Y
[18]	CWE-676	Use of Potentially Dangerous Function	N
[19]	CWE-327	Use of a Broken or Risky Cryptographic Algorithm	N
[20]	CWE-131	Incorrect Calculation of Buffer Size	Y
[21]	CWE-307	Improper Restriction of Excessive Authentication Attempts	N
[22]	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')	N
[23]	CWE-134	Uncontrolled Format String	Y
[24]	CWE-190	Integer Overflow or Wraparound	Y
[25]	CWE-759	Use of a One-Way Hash without a Salt	N

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AFRL/RI .....Air Force Research Laboratory, Information Directorate
API…… ...Application Programming Interface

ASIC……..…… Application-Specific Integrated Circuit
CAD ..... Computer Aided Design
COTS .....Commercial Off The Shelf
CPU..... Central Processing Unit
CRASH ..... Clean-Slate Design of Resilient, Adaptive, Secure Hosts
DARPA..... Defense Advanced Research Projects Agency
DIFT..... Dynamic Information Flow Tracking
DMA ..... Direct Memory Access
DoD..... Department of Defense
FPGA .....Field Programmable Gate Array
HDL ..... Hardware Description Language
IARPA..... Intelligence Advanced Research Projects Agency
LoC .....Lines of Code
MILS .....Multiple Independent Levels of Security
OS .....Operating System
RTEMS .....Real Time Executive for Multiprocessor Systems
RTL .....Register Transfer Level
STA ..... Security Tagged Architecture
STACD .....Security Tagged Architecture Co-Design
STI..... Security Tagged Interconnect
STMP .....Security Tagged Multicore Processor
ST-ZKOS .....Security Tagged Zero Kernel Operating System
TIARA ..Trust-management, Intrusion-tolerance, Accountability and Reconstitution Architecture
TMU.....Tagged Management Unit
VHDL ..... VHSIC Hardware Description Language
ZKOS .....Zero-Kernel Operating System